

Universidade Técnica de Lisboa

Instituto Superior Técnico

Instituto de Telecomunicações – Pólo de Lisboa

Relatório Técnico

*“Pattern Matching Image Compression:
Estudo e implementação”*

Maio 2000

Artur Ferreira

ÍNDICE

1. OBJECTIVOS.....	3
2. INTRODUÇÃO.....	3
3. DESCRIÇÃO DA TÉCNICA <i>PMIC</i>	4
3.1 BASE DE DADOS (DICIONÁRIO).....	4
3.2 ANÁLISE DO PROCESSO DE CODIFICAÇÃO/DESCODIFICAÇÃO	5
3.2.1 <i>Pesquisa de prefixo sobre a base de dados</i>	5
3.2.2 <i>Critério de distorção</i>	6
3.2.3 <i>Exemplo de codificação</i>	8
3.3 ALGORITMO DE COMPRESSÃO.....	9
3.4 FORMATO DO CÓDIGO E ALGORITMO DE DESCOMPRESSÃO	10
3.4.1 <i>Comprimento do código</i>	11
3.4.2 <i>Algoritmo de descompressão</i>	11
3.5 OPTIMIZAÇÕES.....	12
4. IMPLEMENTAÇÃO SEGUNDO O PARADIGMA <i>OBJECT-ORIENTED</i>	12
4.1 MODELAÇÃO.....	12
4.2 RESULTADOS OBTIDOS.....	14
4.2.1 <i>Compressão de imagem</i>	14
4.2.2 <i>Restauro de imagem</i>	16
5. CONCLUSÕES.....	17
6. BIBLIOGRAFIA E REFERÊNCIAS	18
7. ANEXO A – REPRESENTAÇÃO GRÁFICA DA MODELAÇÃO <i>OBJECT-ORIENTED</i>.....	18

1. Objectivos

O presente trabalho tem como objectivos o estudo e a implementação da técnica de compressão de imagem denominada por *Pattern Matching Image Compression* (PMIC). O trabalho é constituído por três partes:

- estudo e apresentação da técnica PMIC, enquadrada com o processo da codificação universal de fonte baseada em dicionário;
- descrição da modelação *object-oriented* e sua implementação na linguagem C++;
- aplicação da técnica PMIC para compressão e restauro de imagem.

Verifica-se a funcionalidade desta técnica de compressão (*proof of concept*), as suas virtudes e defeitos, através da comparação com métodos conhecidos baseados em transformada.

No que se refere à implementação, existem optimizações do método PMIC, constituindo variantes da proposta original. Dado o enquadramento do trabalho, a exploração dessas variantes não faz parte dos objectivos a atingir. Contudo a implementação é suficientemente genérica para permitir a sua posterior inclusão.

2. Introdução

A técnica *Pattern Matching Image Compression* (PMIC) [1] é um método de compressão com perdas aplicado a imagem, criado a partir da compressão baseada em dicionário proposta por Ziv-Lempel [2], mais concretamente do algoritmo LZ77. A compressão baseada em dicionário teve a sua génese na década de 70, tendo sido demonstrada a sua optimalidade [3], relativamente ao conceito de entropia introduzido por Shannon [4]. Hoje em dia existem inúmeras aplicações comerciais que implementam técnicas baseadas em dicionário e suas variantes (são exemplos: *compress* do UNIX, *gzip* e *pkzip*).

A técnica PMIC consiste na utilização da codificação baseada em dicionário e não usa qualquer transformada, tal como a maioria das técnicas de compressão de imagem [1] [5]. A tradicional compressão baseada em dicionário utiliza a pesquisa de *matches* exactos sobre o dicionário, procurando codificar o texto através do maior *match*, usando o menor número possível de *bits*. A aplicação da compressão Ziv-Lempel a imagem surge de forma natural, adicionando a possibilidade de efectuar a compressão com perdas (*lossy* Ziv-Lempel), fazendo com que o algoritmo apresente relativamente baixa complexidade computacional.

A compressão com perdas consiste na pesquisa (segundo dado critério) de um *match* que aproximadamente ocorra no dicionário, tal como descrito adiante.

3. Descrição da técnica *PMIC*

Nesta secção apresenta-se a descrição detalhada da técnica *PMIC*, assentando sobre a sobejamente conhecida compressão de Ziv-Lempel, segundo o algoritmo LZ77. A técnica *PMIC* introduz as seguintes características:

- aplicação a imagem monocromática (com níveis de cinzento);
- compressão com perdas, através da introdução de um nível de distorção.

O modelo geral do processo de compressão [6] é constituído pelas entidades Compressor e Descompressor que englobam duas outras entidades:

- codificador/descodificador;
- modelo.

O modelo detém informação estatística sobre os símbolos produzidos pela fonte. O codificador atribui o código correspondente a cada símbolo ou conjunto de símbolos, segundo a informação obtida a partir do modelo.

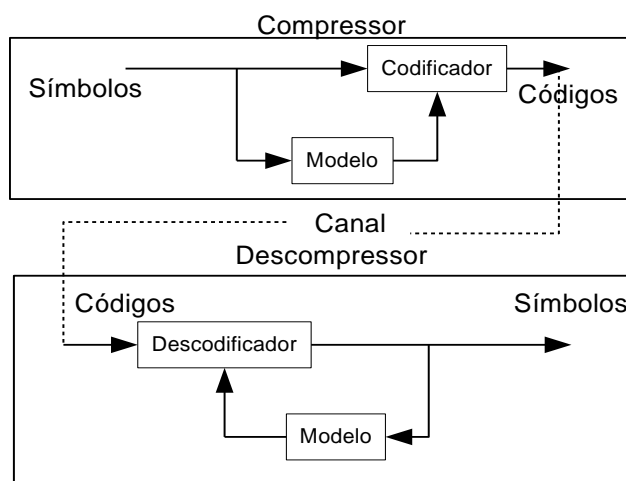


Figura 1 – Modelo teórico do processo de compressão/descompressão.

3.1 Base de dados (dicionário)

A compressão de texto pela técnica baseada em dicionário assenta na pesquisa de prefixos desse texto sobre um conjunto de símbolos (que se caracteriza por ser adaptativo, ou seja, actualizado ao longo da compressão/descompressão) que se designa por dicionário (que corresponde ao modelo da figura 1). Sobre o dicionário pesquisa-se o maior prefixo exacto do texto que se pretende codificar, tendo desta forma uma compressão sem perdas.

Quando se estende este conceito para a compressão com perdas (aplicado a imagem) o dicionário passa a designar-se por base de dados, dado que não se pesquisa por prefixos exactos. A pesquisa do prefixo mais aproximado é feito sobre a base de dados. Esta é

constituída por uma sequência de símbolos, gerada inicialmente antes dos processos de compressão e descompressão. A sua criação é feita segundo a lei de probabilidade¹ designada por P, que deve estar relacionada com a estatística da imagem que se pretende comprimir, de forma a maximizar a eficácia da compressão. A figura 2 mostra um estado possível da base de dados, constituída por diversos valores de pontos da imagem.

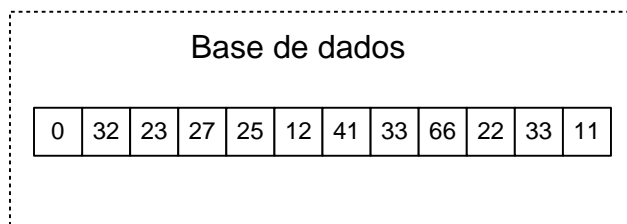


Figura 2 – Representação gráfica de possível base de dados.

Note-se que ao longo do processo de compressão/descompressão a base de dados contém os últimos pontos codificados/descodificados.

3.2 Análise do processo de codificação/descodificação

A codificação de uma sequência consiste em representar a mesma com o mínimo número de *bits*, através de código de dimensão variável na forma (*posição*, *comprimento*). Pretende-se que a construção do código seja computacionalmente eficiente. A maior complexidade de todo o processo de codificação consiste na pesquisa sobre a base de dados, justificando que o tempo de compressão seja dezenas de vezes superior ao tempo de descompressão. A complexidade do processo de descodificação é mínima dado que o mesmo consiste na leitura de códigos e escrita de símbolos.

3.2.1 Pesquisa de prefixo sobre a base de dados

Estando perante um processo de codificação baseado em dicionário, colocam-se dois problemas fundamentais:

- pesquisa eficiente sobre a base de dados;
- encontrar o maior prefixo que mais aproxima o texto a comprimir (*parsing*).

O primeiro problema coloca-se do ponto de vista computacional, dado que se pretende pesquisa rápida. As estruturas de dados devem estar implementadas para minimizar o tempo de pesquisa. No presente trabalho não houve a preocupação de otimizar as estruturas de dados para minimizar o tempo de pesquisa, dado que o objectivo é realizar a implementação para verificar a funcionalidade do conceito (*proof of concept*). Existem técnicas baseadas na estrutura em árvore que minimizam o tempo de

¹ Por exemplo a lei de probabilidade P pode ser uma cadeia de Markov, de 1ª ou 2ª ordem.

pesquisa sobre o dicionário [9]. A representação do dicionário na forma de *array* de listas também é uma forma de reduzir o tempo de pesquisa.

O segundo problema caracteriza-se pelo tipo de *parsing* efectuado sobre a base de dados. Uma vez que se está a efectuar compressão com perdas, aplica-se a técnica *greedy parsing*, obtendo o primeiro prefixo que respeita o critério de distorção, critério este que se apresenta na secção seguinte.

3.2.2 Critério de distorção

A escolha do prefixo mais aproximado a determinada sequência é feita segundo o valor do erro quadrático. Desta forma apenas se aceitam prefixos que têm erro quadrático não superior ao valor máximo permitido.

Para evitar situações pontuais de grande diferença entre pontos da sequência e da base de dados, adicionou-se a restrição Δ designada por máxima diferença que consiste em aceitar um prefixo apenas quando todos os seus elementos têm afastamento inferior ou igual a determinado valor pré-estabelecido Δ . A medida de distorção é dada expressão (1).

$$\begin{aligned}
 &x, \text{ sequência a codificar} \\
 &y, \text{ base de dados} \\
 d(x_i, y_i) &= \begin{cases} +\infty, & \text{se } |x_i - y_i| > \Delta \\ E = (x_i - y_i)^2 & \end{cases} \quad (1)
 \end{aligned}$$

A figura 3 mostra a título de exemplo o cálculo da distorção de duas sequências:

- a sequência 1 que obedece ao critério do erro quadrático, mas que falha segundo o critério da máxima diferença;
- a sequência 2 que apresenta um erro quadrático superior mas que respeita o critério da máxima diferença.

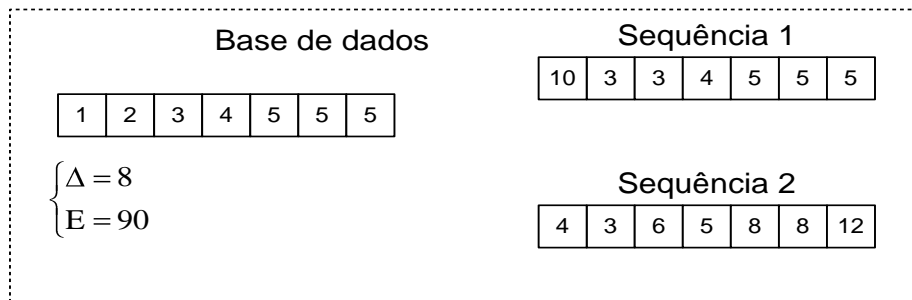


Figura 3 – Exemplo de cálculo da distorção entre a base de dados e duas sequências.
 Nota: A base de dados e as sequências têm a mesma dimensão apenas para simplificar o raciocínio.

O cálculo da distorção para as duas sequências produz os valores apresentados na tabela I, onde se verifica que apenas a sequência 2 tem distorção aceitável, apesar do maior erro quadrático.

	Sequência 1	Sequência 2
Erro quadrático	82	87
Máxima diferença	9	7

Tabela I – Cálculo da distorção de duas sequências sobre a base de dados.

Na figura 4 apresenta-se o algoritmo completo para a pesquisa de determinado prefixo sobre a base de dados. Note-se que existem versões otimizadas que podem ser encontradas em [1].

Prefix

Entrada: x_1^n , base de dados (com 'n' símbolos).
 y_1^m , sequência a comprimir (com 'm' símbolos).

Saída: Código (t,k), tal que para a posição t, tem-se que k é o maior inteiro que respeita a condição de distorção $d(x_t^{t+k+1}, y_1^k) \leq D$.

Início

Iniciar $S_{ij}=0, \forall i, j \quad i \in [1, n-m] \quad j \in [1, m]$

for i=1 to n-m do
 for j=1 to m do
 $S_{ij} = S_{ij-1} + d(x_{i+j-1}, y_j)$
 end
end

k = maior j tal que $S_{ij} < jD$.
t = i correspondente ao maior j.
Retornar (t,k).

Fim

Figura 4 – Algoritmo de pesquisa de um prefixo na base de dados.

Sobre a expressão (1) importa referir que a componente do erro quadrático pode ser substituída pela distância de *Hamming*, definida [1] de acordo com o seguinte critério:

$$E(x_i, y_i) = \begin{cases} 0, & \text{se } x_i = y_i \\ 1, & \text{caso contrário} \end{cases} \quad (2)$$

3.2.3 Exemplo de codificação

A figura 5 exemplifica a codificação de uma sequência sobre determinada base de dados, onde se constata a obtenção de *match* parcial (que respeita o critério de distorção). Também se ilustra a situação em que três símbolos são codificados em claro conduzindo à actualização da base de dados. Note-se que apenas os símbolos que passam em claro é que são inseridos na base de dados, de forma a não aumentar a degradação no processo de descompressão.

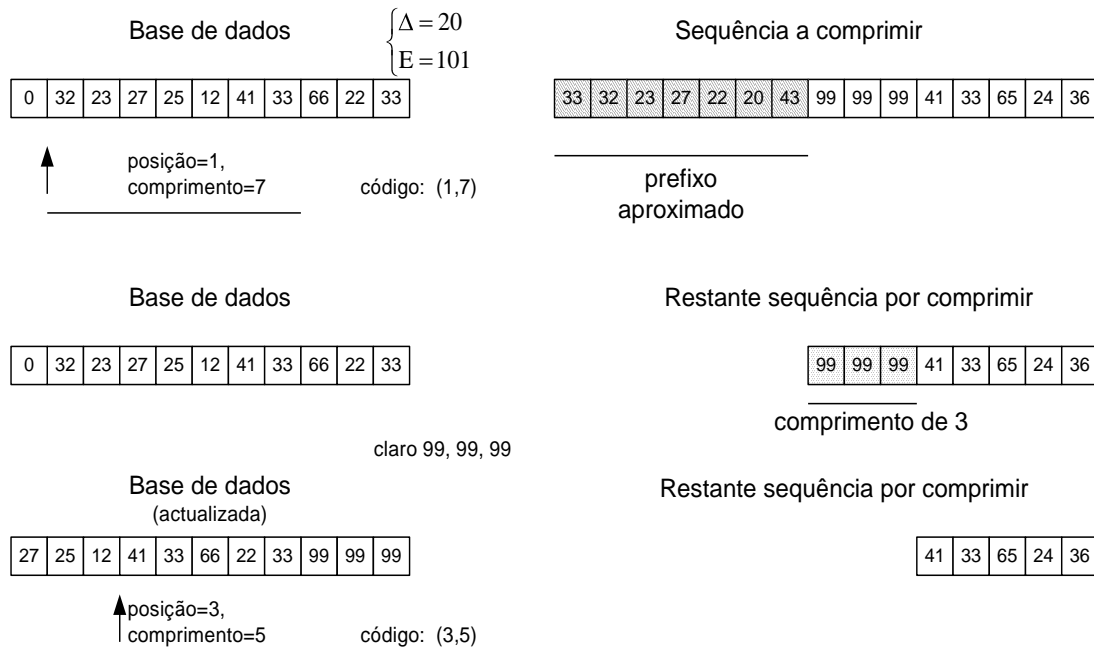


Figura 5 – Exemplo de compressão: símbolos em claro e codificados.

Sobre a base de dados exemplo da figura 5, a compressão da sequência produz o seguinte código:

(1,7) 99 99 99 (3,5)

Considerando uma imagem $N \times N$, a base de dados consiste nas últimas f (valor inteiro) linhas anteriores à linha corrente, sendo actualizada à medida que se avança no processo de codificação, criando o efeito de janela deslizante (*Sliding Window*), tal como sugerido em [2] e representado na figura 5. Sobre uma imagem de 512×512 considerando quatro linhas na base de dados tem-se que a mesma tem dimensão 2048 bytes (2 Kb), levantando problemas de eficiência na pesquisa.

A figura 6 ilustra a descodificação da sequência produzida no exemplo anterior.

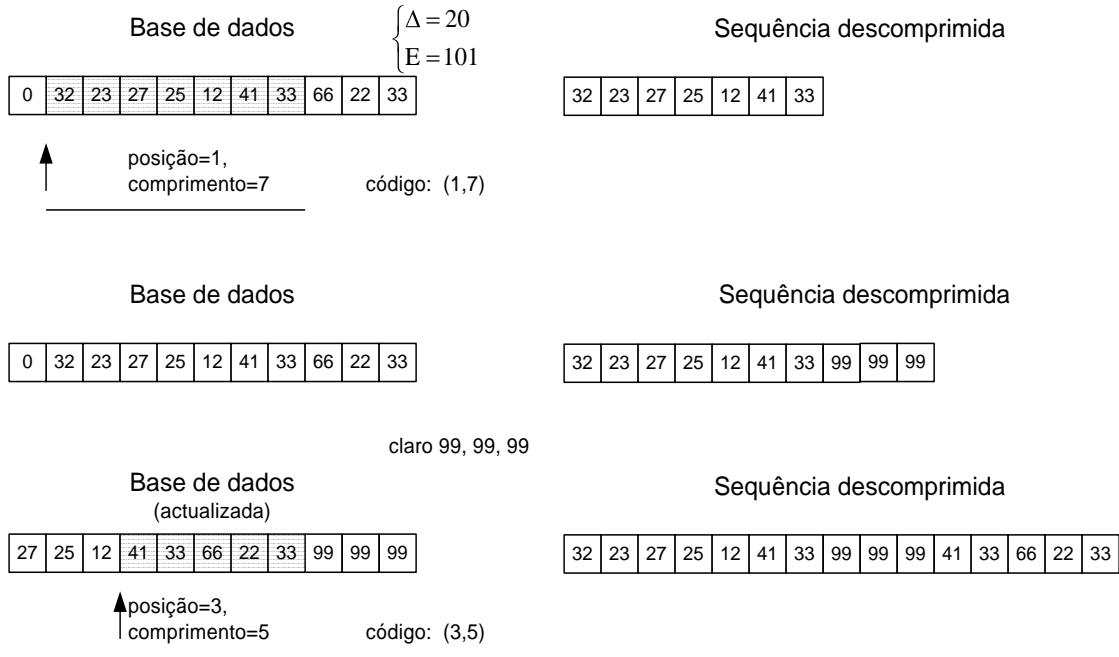


Figura 6 – Exemplo de descompressão: símbolos em claro e codificados.

Pela análise da figura 6, efectuando a comparação com a figura 5, confirma-se que o processo de descodificação é bastante menos complexo que a codificação, na medida em que apenas se efectuam leituras, consultas à base de dados e escritas.

3.3 Algoritmo de compressão

O processo de compressão é efectuado de forma sequencial. Cada linha da imagem é comprimida individualmente de forma sequencial da esquerda para a direita. O processamento de uma linha completa é apresentado na figura 7.

Compress_row
 Analisar sequencialmente a linha desde o início (da esquerda para a direita):

- pesquisar na base de dados (através do algoritmo *prefix*) pelo maior prefixo que codifica a maior porção possível da linha;
- se a dimensão do prefixo for superior a determinado valor fixo então codifica-se na forma (*posição,comprimento*); caso contrário escreve-se o prefixo em claro, com a dimensão igual ao valor mínimo definido.

Figura 7 – Algoritmo de compressão de linha.

A análise da dimensão do prefixo obtido na pesquisa sobre a base de dados é um parâmetro importante a ter em conta no processo de codificação dado que existe perda de eficácia de compressão quando se codificam prefixos com baixo número de símbolos (geralmente igual ou inferior a quatro). Esta ideia de enviar símbolos em claro foi introduzida por Storer e Szymansky [7], na variante do LZ77 denominada por LZSS, que tem vindo a ser aplicada na compressão sem perdas (*lossless*) [9]. O algoritmo para a compressão da imagem resume-se a evocar repetidamente o algoritmo *compress_row* para todas as linhas.

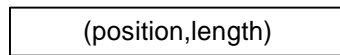
3.4 Formato do código e algoritmo de descompressão

Dadas as duas possibilidades de codificação de símbolos em claro ou através do par (posição, comprimento) torna-se necessária a introdução de mais informação no código gerado, nomeadamente um dígito binário a indicar a forma de codificação. Existem optimizações ao processo de codificação (referidas na secção 3.5) que quando usadas implicam a extensão do formato do código aqui apresentado. A figura 8 apresenta o formato do código utilizado para a compressão das linhas. Na implementação efectuada optou-se por utilizar um formato ligeiramente diferente do proposto em [1].

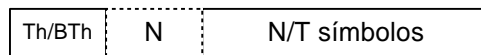


O primeiro campo do código é constituído pelo *bit* C/P (Code/Pointer) que tem o seguinte significado:

- com o valor 1, indica *code* e o segundo campo tem o seguinte formato:



- com o valor 0, indica *plain* e nesta situação o segundo campo toma outro formato:



Th/Bth: 1-Threshold , o campo N não existe e seguem-se *NThreshold* símbolos;

0-Below Threshold, o campo N tem significado;

N: (opcional) número de símbolos que se seguem se *Th/Bth*=0;

N/T símbolos: conjunto de símbolos em claro;

Figura 8 – Formato do código PMIC.

3.4.1 Comprimento do código

O comprimento do código (*posição, comprimento*) pode ser calculado da seguinte forma: considerando que a base de dados tem comprimento de L_n símbolos, o código para a posição e para o comprimento necessita de ter o número de dígitos binários dados pela expressão:

$$L = \text{ceil} (\log_2 (L_n)) \quad (3)$$

Nota: o operador ceil significa o arredondamento para o próximo inteiro igual ou superior.

Conclui-se que no total para efectuar a representação do par (*posição, comprimento*) são necessários $2L$ dígitos binários. Desta forma conclui-se também que sempre que o comprimento do prefixo for inferior a $2L$, é vantajoso enviar o prefixo em claro. O cálculo da fronteira da dimensão do prefixo foi levado em linha de conta na implementação.

3.4.2 Algoritmo de descompressão

O conjunto de acções efectuado pelo algoritmo de descompressão pode ser observado na figura 9.

```

Decompress
Enquanto não atingir o final do ficheiro comprimido:
  Ler o dígito binário C/P
  if (C/P=Code)
    Ler (t,k);
    Obter a sequência na posição t com comprimento k;
  else
    if (Th/Bth= Bth)
      Ler o valor do número de símbolos N;
      Ler os N símbolos em claro;
    else
      Ler os Nthreshold símbolos em claro;
    end
    Actualizar a base de dados com os símbolos em claro;
  end
  Escrever a sequência no ficheiro de destino;
    
```

Figura 9 – Algoritmo de descompressão.

3.5 Optimizações

Existe um conjunto de optimizações e variantes [1] que podem ser aplicadas sobre o processo de compressão PMIC. Entre as mesmas destacam-se:

1. prefixos invertidos;
2. prefixos na forma deslocada e somada;
3. *run length encoding* para as zonas da imagem com pequena variação;
4. utilização de níveis de distorção variáveis e adaptativos.

A utilização destas variantes implica a extensão do formato do código apresentado na secção 3.4, tornando-se necessária a adição de mais informação no final do mesmo.

4. Implementação segundo o paradigma *object-oriented*

O desenho do compressor e descompressor PMIC segundo o paradigma da programação orientada por objectos deve obedecer às seguintes características essenciais:

- ser independente da linguagem de programação utilizada;
- reflectir todas as características descritas sobre a técnica *PMIC*;
- os pontos de expansão devem estar perfeitamente identificados, para facilitar a futura inclusão de novas técnicas e optimizações.

Tendo em conta estes aspectos essenciais, foi desenhada a modelação para resolver o problema, que se apresenta na secção seguinte.

4.1 Modelação

Analisando a descrição do compressor PMIC efectuada até este ponto do texto, verifica-se que este utiliza uma base de dados, que pode ser identificada como o modelo utilizado no processo de codificação. Por cada linha (sequência) a codificar é feita a consulta sobre a base de dados, a qual responde com informação sobre o maior prefixo encontrado. Internamente, o compressor detém controlo sobre a sub-entidade que efectua a codificação, ou seja, a atribuição do código à sequência de símbolos. No processo de descompressão tem-se um mecanismo simétrico de consulta e actualização da base de dados.

A base de dados é essencialmente constituída por uma sequência de símbolos, com dimensão fixa. Deve fornecer os mecanismos de consulta e actualização nos processos de compressão e descompressão, tal como evidenciado nos exemplos das figuras 5 e 6. No entanto na compressão existe maior complexidade na base de dados, uma vez que esta deve ainda fornecer a funcionalidade de pesquisa de prefixos segundo o critério de distorção dado pela expressão (1). A criação da base de dados é feita segundo

determinada lei de probabilidades P . A figura 10 ilustra as entidades que representam a base de dados, nos processos de compressão e descompressão.

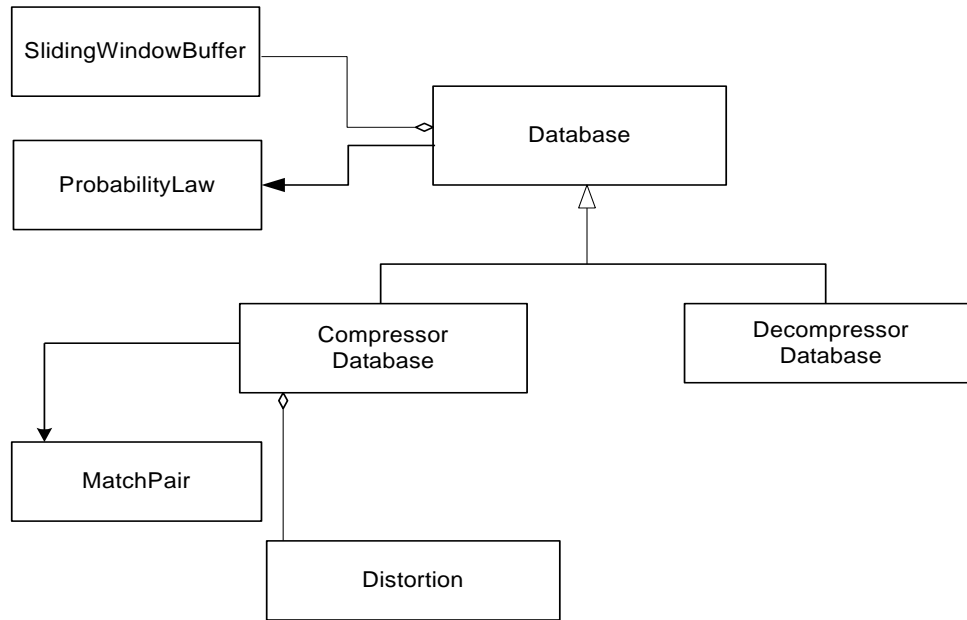


Figura 10 – Modelação da base de dados para compressão e descompressão.

Por sua vez o compressor e descompressor utilizam as componentes da base de dados adequadas ao seu funcionamento, tal como ilustrado na figura 11.

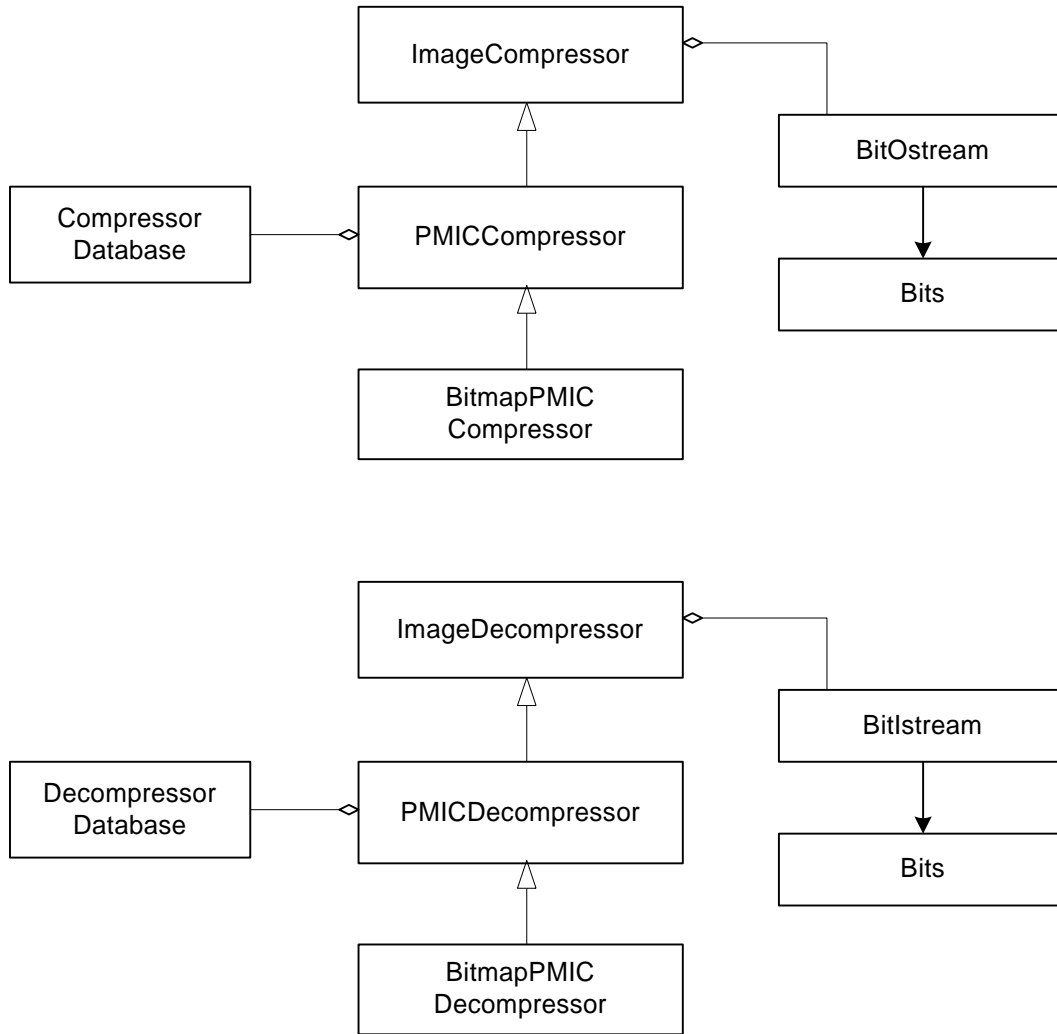


Figura 11 – Modelação das entidades compressor e descompressor. Relação com a base de dados.

4.2 Resultados obtidos

Nesta secção apresentam-se alguns resultados obtidos na compressão e no restauro de imagens. Os resultados referem-se à imagem de teste “Lena”. Foram efectuados testes sobre outras imagens. Dado que essas imagens não são *standard*, optou-se por não se apresentar esses resultados.

4.2.1 Compressão de imagem

Os resultados apresentados na tabela II referem-se a uma base de dados constituída por 5 linhas com 10 símbolos cada (dimensão de 50 símbolos). A dimensão da base de

dados é um factor importante a ter em conta dado que determina directamente o comprimento dos códigos, segundo a expressão (3).

Parâmetros da distorção	Percentagem removida
E=100 Δ=12	21.9
E=100 Δ=15	25.8
E=150 Δ=20	31.2
E=350 Δ=25	38.5
E=550 Δ=35	44.4

Tabela II – Taxas de compressão obtidas para diferentes níveis de distorção.

A percentagem removida [9] é obtida segundo a expressão (4).

$$\text{removed} = 100 \left(1 - \frac{\text{compressedSize}}{\text{originalSize}} \right) \quad (4)$$

A figura 12 mostra a imagem original do lado esquerdo e a imagem obtida após descompressão, na última situação de teste apresentada na tabela, onde se pode verificar que o ficheiro comprimido tem dimensão 44.4% inferior à dimensão original. Verifica-se distorção na imagem da direita. A distorção nota-se essencialmente ao longo das linhas da imagem, porque a compressão das linhas é sequencial.



Figura 12 – Imagem de teste: original e após compressão com perdas.

Outra experiência efectuada, que não deu resultados satisfatórios em termos de taxa de compressão consistiu na aplicação de *Run Length Encoding* sobre a imagem antes de aplicar o algoritmo *PMIC*.

4.2.2 Restauro de imagem

A aplicação da compressão com perdas também pode ser efectuada com o objectivo de efectuar restauro de imagem deteriorada. Sobre a imagem de teste “*Lena*” adicionou-se ruído impulsivo, com diferentes frequências de ocorrência. Em seguida aplicou-se a compressão e descompressão com perdas ($E=250$ $\Delta=15$) e comparou-se a imagem original com a imagem descomprimida. A figura 13 mostra os resultados obtidos nas operações de restauro.



Figura 13 – Resultados da aplicação do PMIC para restauro de imagem.

Do lado esquerdo apresenta-se a imagem contaminada e do lado direito a imagem obtida após o processo de compressão/descompressão com perdas ($E=250 \Delta=15$). Na primeira situação a imagem esquerda tem 10% de ruído. Na segunda situação tem-se 25% de ruído. Verifica-se o funcionamento do processo de compressão e descompressão com perdas para restauro de imagem.

5. Conclusões

A aplicação da codificação Ziv-Lempel com perdas surge como extensão da aplicação da ideia original da compressão sem perdas. Em relação aos métodos baseados em transformada tem-se a desvantagem de não ser possível determinar com exactidão quais as consequências da introdução de perdas. Por exemplo não se consegue determinar ou prever a diferença entre os espectros (análise em frequência) da imagem original e da imagem após descompressão. Verifica-se ainda que este processo de codificação é mais lento [1]. De facto, o cálculo de transformadas tais como a DCT (*Discrete Cosine Transform*) e a FFT (*Fast Fourier Transform*) é cada vez mais eficiente não só pela elevada velocidade de processamento dos computadores hoje em dia, mas essencialmente pelo facto da implementação das mesmas estar adaptada à tecnologia de construção dos processadores, como por exemplo o MMX da Intel.

É expectável que a aplicação do método PMIC obtenha melhores resultados se for aplicado aos coeficientes de uma transformada aplicada à *priori* sobre a imagem, tal como por exemplo a DCT. A figura 14 mostra o diagrama de blocos do compressor resultante.

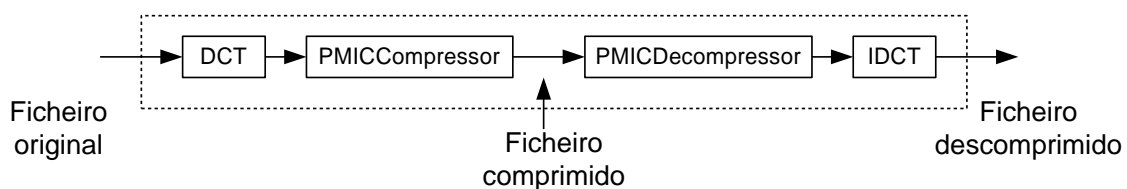


Figura 14 – Diagrama de blocos da compressão PMIC com aplicação prévia da DCT.

Verificou-se ainda que a compressão com perdas serve para efectuar restauro de imagem. A extensão do método para 2D [8] é uma técnica que promete melhorias nos resultados, dado que a distorção deixa de se verificar ao longo das linhas. Juntamente com a aplicação de uma transformada à *priori*, estes são dois possíveis pontos de expansão do presente trabalho.

6. Bibliografia e Referências

- [1] M. Atallah, Y. Génin, W. Szpankowski, “Pattern Matching Image Compression: Algorithmic and Empirical Results”, IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 21, n°7, pp. 614-627, July 1999.
- [2] J. Ziv, A. Lempel, “A universal algorithm for sequential data compression”, IEEE Transactions on Information Theory, vol. 23, n°3, pp. 337-343, May 1977.
- [3] J. Ziv, A. Lempel, “Compression of individual sequences via variable-rate coding”, IEEE Transactions on Information Theory, vol. 24, n°5, pp. 530-536, September 1978.
- [4] C. Shannon, “A Mathematical Theory of Communication”, Bell System Technical Journal, Vol. 27, pp. 379-423, pp. 623-656, July, October, 1948.
- [5] A. K. Jain, **Fundamentals of Digital Image Processing**, Prentice Hall, 1989.
- [6] J. Rissanen, G. Langdon, “Universal Modeling and Coding”, IEEE Transactions on Information Theory, vol. 27, n°1, pp 12-23, January 1981.
- [7] J. Storer, T. Szymansky, “Data Compression via textual substitution”, Journal of ACM, vol. 29, n°4, pp. 928-951, October 1982.
- [8] M. Alzina, W. Szpankowski, A. grama, “2D-Pattern Matching Image and Video Compression: Preliminary Results”, November 7, 1998.
- [9] M. Nelson, “The Data Compression Book”, Second Edition, M&T Books, 1996.

7. Anexo A – Representação gráfica da modelação *object-oriented*